Chapter 5

Advanced Particle Filter Methods

Roi Yozevitch and Boaz Ben-Moshe

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.69236

Abstract

This chapter presents a set of algorithmic methods based on particle filter heuristics. We start with an introduction to particle filters, which covers the main motivation and related works. Then, the generic framework for particle filter algorithm is presented, followed by two important use cases regarding indoor positioning and multitarget tracking; for both problems, modified particle filter algorithms are presented followed by experimental results, implementation remarks, and a discussion. Finally, a short list of conclusion and future work are presented.

Keywords: particle filter, localization, navigation heuristics, position accuracy estimation, GNSS jamming, indoor navigation, multitarget tracking

1. Introduction

Among the heuristic techniques and filters available to the researcher, the "particle filter" technique is one of the most flexible to use. Unlike the Kalman filter on its variations (especially the Extended and Unscented Kalman filters, namely EKF and UKF), the particle filter is neither restricted to Gaussian a posterior distribution nor to a unimodal solution. The number of problems that can be described by a particle filter are numerous. Why is particle filter considered a heuristic technique? The answer is obvious: In essence, the particle filter is a nonparametric way to sample the desired probabilistic space. Real-world probabilistic spaces are often complex and cannot rely upon Gaussian/convex assumptions. Sampling such an arbitrary space utilizing finite numbers of particles is naturally an approximation. Hence, the particle filter should be considered as a generic heuristic technique for modeling complex probabilistic spaces.

1.1. Motivation

In the last decade, a massive amount of research has been devoted to both autonomous cars and unmanned aerial vehicles (UAVs). Part of this research includes multiagent localization (e.g., swarms of robots). The contemporary autonomous car is equipped with a wide range of sensors which requires an intelligent data fusion to construct a reliable reality map. One must remember that autonomous cars are a mission critical system; that is, the system must work all of the time. Such system cannot solely rely on global positioning system (GPS), since it is occasionally not available (e.g., dense urban canons and city tunnels). Furthermore, in those systems, estimating the error bounds is not just a convenient feature. In December 2011, an American UAV landed in a hostile enemy environment, over 200 km from its original route. This drone was equipped with the state-of-the-art military navigation system. This example (later known as Iran-U.S. RQ-170 incident [1]) demonstrates the crucial importance for modeling the error space based on data from several sensors.

While fully autonomous cars are still a far vision, current new cars are required to have active-safety systems in order to get a full safety rating (five-star safety level). Those systems are capable to detect the car position in the lane and warn the driver about pedestrians and other cars.

Particle filters can address all of the above challenges. In this chapter, we present new algorithmic improvements to estimate the system's error bounds, to localize and track multiagents and more. Rapid advances in hardware acceleration and the intense use of parallel computing (e.g., graphics processing unit, GPU) across the board are perfect for implementing the particle filter in many scenarios.

In this chapter, we focus on two abstract problems that the state-of-the-art particle filters are dealing with. The first problem is how to reduce the number of particles without losing (a lot) of knowledge. The second problem is how to both localize and track more than one agent simultaneously. Those abstract algorithmic challenges are demonstrated via two case studies.

1.2. Related works

The concept of simulating particles started in the 1960s [2] to address nonlinear parabolic partial differential equations arising in fluid mechanics. The term "particle filter" was first defined in the mid 1990s [3]; however, other researchers have used the terminology of "sequential Monte Carlo" [4]. In the last two decades, a considerable amount of research work was devoted to develop various heuristics based on particle filters [5–8]. Positioning algorithms are often based on particle filters as it is a robust method for performing sensor fusion [9, 10]. In order to improve the expected runtime performance, several researchers have developed a GPU-based parallel-methodology for particle filters [11–13]. It is important to note that a particle filter is a generic technique and is being used to solve (approximate) a wide range of nonlinear problems including video stabilization, tracking, and multitarget tracking (MTT) [14, 15]. Finally, particle filters are becoming a common high-level heuristic which is being used in many autonomous robotic applications [16–18] allowing both robustness and improved error-bound estimations.

2. Particle filter: general heuristic

As stated above, the particle filter is a member of the nonparametric multimodal Bayesian filters family. In the particle filter case, the posterior is estimated by a finite number of parameters (particles). These particles are represented as $\chi_t := x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[N]}$, where N is the number of particles. A belief function, $bel(x_t)$, is evaluated for each particle. This function serves as the particle's weight (importance). Thus, $\{x_t^{(K)}, (w_t^{(K)},) : K \in \{1,\ldots., N\}\}$. The importance weight is proportional to the likelihood of a specific particle [19]:

$$x_t^{[K]} \sim p(x_t|z_{1:t}, u_{1:t}) \times w_t^{(K)} \tag{1}$$

In Eq. (1), $z_{1:t}$ and $u_{1:t}$ are the sense and action functions, respectively. The action function u_t moves the particles at time t. The movement is usually derived from odometry (wheel encoding, pedometers). A fundamental characteristic of the action function is the accuracy reduction of each particle. Since any real-world movement contains (to a certain degree) noise, moving a particle **decreases** certainty regarding its state. The sense function, z_t, does exactly the opposite. This function evaluates the particle's likelihood based on its sensors (vision, Inertial Measure Unit - IMU, etc.). Assuming the particle measures distance from n known landmarks, one can evaluate the weight of each particle p as:

$$z(x_t) \sim \sum_1^n f(x|\mu, \sigma^2) \tag{2}$$

where f represents the Gaussian function:

$$f(x|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{3}$$

The value x represents the measured distance to the nth landmark and μ represents the theoretical distance between the known landmark and the particle position. The closer a particle to its true position, the value $abs(x - \mu)$ diminishes and the particle's weight increases. Thus, the sensor function **increases** certainty regarding its state. This process is also called Monte Carlo localization (MCL) [19].

The particle filter is a Bayesian, multimodal, nonparametric estimation tool. In order to fully understand this concept, several terms must be clarified:

- The Bayesian property implies that the estimation at t_1 is derived from the estimation at t_0.
- The multimodal feature implies that unlike Kalman filters, there can be more than a single plausible solution.
- The nonparametric property implies the posterior is not restricted to a Gaussian (or other parametric) probability density function (PDF).

Data: Sensor data, Map (optional)
Result: Improved position
1. Init: Distribute a set P of M particles in the ROI;
2. Estimate the velocity vector $\vec{v}_t$ from the sensor measurements (e.g., gyroscope, encoder, pedometer…);
3. **for each** *particle* $p \in P$ **do**
   4. Approximate the **action function** $u_t(p)$ from $\vec{v}_t$;
   5. Update p-position according to $u_t(p)$;
   6. Evaluate the belief (weight) function: $w(p)$ based on the known landmarks and map restrictions (Map Matching);
7. Resample P according to the likelihood (weight) of each particle;
8. Compute best position in $P \in ROI$ as $pos(P)$;
9. **Report** $pos(P)$;
10. Approximate the error estimation $err(P)$;
11. if (err(P) < ROI) go to 2 else go to 1;

Algorithm 1: Simple Particle Filter Algorithm.

Properties 2 and 3 mean that the PDF is constructed by the particles themselves, therefore, any arbitrary function can be described that way—there is not a single best estimate. A particle can be in position A or B with equal likelihood. Thus, a particle filter is well suited for scenarios where one needs to estimate its location within a building with two (or more) identical rooms.

Each particle is a state vector (similar to the Kalman Filter) with a dimension n. Most localization problems define each particle to a set of a position and a velocity vectors:

$$x_t^{[K]} = \left\{ position, \overrightarrow{velocity} \right\} \tag{4}$$

A 2D problem demands for $n = 4$ and a 3D problem demands for $n = 6$. One can also include the particle's orientation (crucial in vehicle localization) and other features as well.

2.1. The generic particle filter algorithm

The algorithm presented is a generic PF localization algorithm. One seeks to find an accurate absolute position from noisy measurements. The action function is usually derived from the odometer (e.g., wheel's encoder).

2.2. Reporting the best solution

As opposed to the traditional Kalman filter where the current state vector is also the filter's best guess, a particle filter holds N different solutions (some more plausible than others). How can one determine the "best" particle? The following are the most common approaches:

- The "best" particle will be computed as the center-of-mass of all particles. This is achieved by averaging all the particles.
- A more sophisticated approach is to compute a weighted average of all particles. Each particle reports its likelihood (weight), so this is relatively straight forward to implement. One should carefully notice that in both methods, the "best" estimate is almost never a

valid particle, only an average of valid particles. When external restrictions exist on the particle themselves, the produced "best estimate" may not obey those restrictions.

- One can simply pick the particle with the highest weight. This is the easiest approach to adopt.
- The first approach can be fused to a single method; choose the particle with the highest weight and compute a weighted center of mass solution around it.

The above methods have one thing in common—they all assume a single solution exists and they seek to find it. In other words, they do not take advantage of the inherent multimodal characteristic of the filter itself. Assuming there are two equally likely solutions, the first two methods will produce a bad guess and the last two will produce only one good solution. Such scenarios do exist when tracking more than one agent and we will discuss those types of scenarios in the following sections.

2.3. Particle filter flaws

While relatively easy to understand and implement, the naive particle filter method suffers from several flaws. As explained above, the naive algorithm is incapable of handling multiagent scenarios. It is true that most localization algorithms seek a single best estimate but this is not true of all of them.

The second point concerns the algorithm runtime complexity. Each particle can be computed independently of the other, thus $O(|N|)$, where $n = |N|$, is the number of particles. Doubling N will double the expected runtime. Alas, the configuration space grows exponentially with respect to the state vector's size. As such, extending a 2D PF to 3D is very complicated.

Finally, as always, real-world scenarios are very different from the sterile PF examples found in the literature. A PF sense function is usually explained utilizing distance from several **known** landmarks where each distance has a Gaussian noise with $\mu = 0$. This scenario does simplify the math but does not describe real-world sensors.

The following section is dedicated to real-world indoor localization problem.

3. Smartphone-based indoor navigation

Contemporary navigation heavily relies on Global Navigation Satellite Systems (GNSS) such as the American GPS and the Russian GLONSS. GNSS signals, however, cannot be used inside buildings. Therefore, indoor navigation depends on other type of sensory data.

3.1. Problem state

Given a smartphone receiver in a well-defined region of interest (ROI), find its most plausible location using only the phone's inherent sensors (IMU, camera, etc.).

The best candidate for this task is the smartphone WiFi module. Each WiFi router (transmitter) sends a signal (also called a WiFi-beacon) at ≈ 10 Hz. Since the WiFi-received signal strength indication (RSSI) decreases with the distance between the router and the phone, one can utilize this information as the conventional landmark. The WiFi scenario differs from the classic (known) landmark sensors in three main aspects:

1. The routers' position is not known. Moreover, in a conventional shopping mall there could be over 100 different WiFi routers. New routers are continuously added and old routers are discarded.
2. The RSSI is not solely affected from the distance. The phone orientation (how the antenna is held), obstruction (walls and the human body), and the building geometry are more important. This means that far signals can be received with a relatively high RSSI while near signals can be received with relatively low RSSI.
3. Although WiFi signals travel at ≈ c (speed of light), the receiver occasionally detects them with a 1–2 second delay. This is due to the way a WiFi scan is performed over the available channels (commonly 13 in 2.4 GHz). At pedestrian speed, 2 s of delay may not be an issue. However, when the user crosses a critical section, this can lead to significant errors.

The above flaws make the sense function hard to intelligently construct. Not only is the RSSI a poor distance indicator, the landmarks (WiFi routers) may be obsolete.

3.2. RF finger printing

In the last two decades, there has been massive research on indoor position, see Refs. [20, 21] for general surveys on indoor positioning technologies. In most cases, some sort of RF finger printing algorithm was implemented [22]. Indoor location services are offered on mobile platforms as Google-Android, Apple-IOS, and Windows-Mobile, and they are commonly based on 3G/4G cellular networks, combined with WLAN fingerprinting (WiFi, Bluetooth (BLE)). The expected accuracy of such systems is commonly 5–30 m at 0.3–1 Hz. In the scope of this chapter, we assume that some kind of location service is available for the user and we present a set of particle filters designed to improve the accuracy and robustness of such services.

3.3. Map constraints

Most navigation applications use an underlying map on which the user position is presented. Such maps can be used by the particle filter algorithms for improving the evaluation of each moving particle with respect to the map constraints (i.e., walls). **Figure 3** presents an example of a floor plan with about 10% walls—yet, using such a map combined with the user movement (action function) allows the particles to converge rapidly. For simplicity, the map constraints are presented here in a binary mode which basically divides the region of interest to areas in which the user can be (white) and restriction zones (black) in which the user may not be. Naturally, one can think of a finer model with several levels or even continuance values. Yet, in most cases the binary model is sufficient and allows for a simple implementation.

3.4. Intelligent weight evaluation

Occasionally, momentary misclassifications will happen due to obstructions (e.g., walls). Hence, one must also consider the particle's history and incorporate it into the weight function. The strengths of this approach are twofold: First, a positive feedback will cause more probable particles to become better in time. Second, this method reduces to minimum the influence of those momentary errors. We set $0<k<1$ to be the ratio between the current estimation and the history weight. A more robust solution will be achieved with lower k. Noisy environments should favor the history. Note that k is the equivalent to the Kalman K gain. Each estimation problem has a different optimal k values:

$$weight(x_t^{[K]}) = k \times sense(x_t^{[K]}) + (1-k) \times weight(x_{t-1}^{[K]}) \tag{5}$$

where $weight(t)$ is the average of the current sense function with all of its history.

Why is it important? Why not increase the number of particles?

Given the number of particles $M \to \infty$, there is a mathematical proof of the convergence of the filter [19]. However, "infinite" number of particles is neither practical nor efficient. Moreover, as stated above, the number of particles grows exponentially with the problem's dimension; one should seek for the minimum optimal number of particles and the rigorous methods for determining that number which are rare. We have used 100, 500, 1000 and 2500 particles throughout the experiments. **Figure 1** demonstrates how the more the particles being used, the smaller the average (steady) error is. We denote "average error" as the average weighed distance between all the particles and the true position. Moreover, as elaborated in Section 3.4, since each particle have no memory of its history, a momentary misclassification can cause significant error as depicted in **Figure 1**. One hundred particles are usually considered as a practical lower bound. Below this number, the solution will often diverge. On the other hand, 5000 particles (and above) produce errors very similar to 2500 particles. In essence, there is a trade-off between the number of particles and the algorithm's accuracy. Although the trade-off is not linear (increasing the number of particles from 100 to 200 will yield much higher improvement than increasing the number of particles from 1000 to 1200), the accuracy is a monotonically increasing function of the number of particles.

One can have both a small number of particles (efficiency) and a very accurate algorithm by implementing the intelligent weight function (Eq. (5)). When the particle's history is taken into account, the errors caused by the small amount of particles are overcome. **Figure 2** proves this thesis. Given a small number of particles (100), the suggested method improves the solution considerably. However, as expected, the improvement is barely noticed for much bigger N values.

What do these graphs mean? As we see, these graphs tell a very interesting story. At first glance, one might suspect this method is not "kosher" since the likelihood of an arbitrary particle was already expressed in the resampling phase—the higher the weight (likelihood) a particle has, the higher the chance for it to be resampled over again. Therefore, incorporating its last weight value seems wrong—creating a deformed unrealistic probability space. However, as clearly demonstrated in **Figure 2**, utilizing this method, especially for a small number

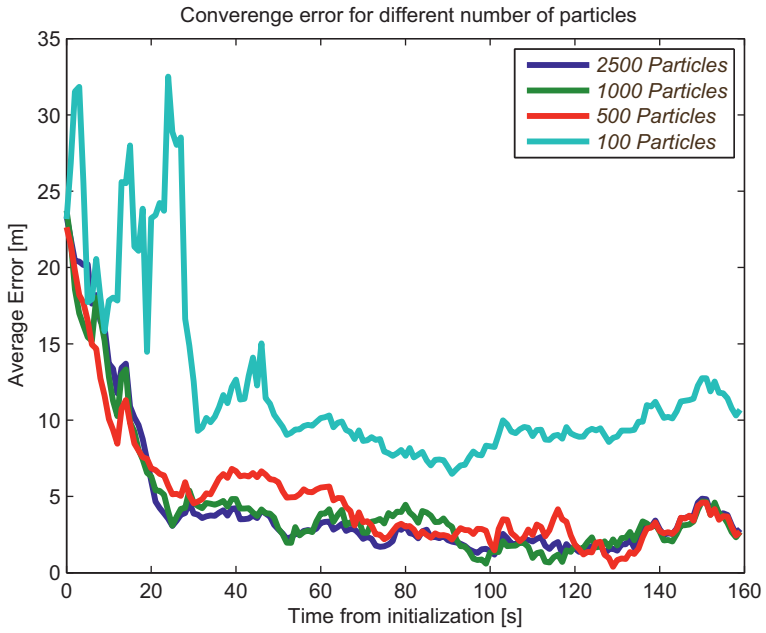

Figure 1. Average error for different numbers of particles. The more the particles being used, the smaller the average error.

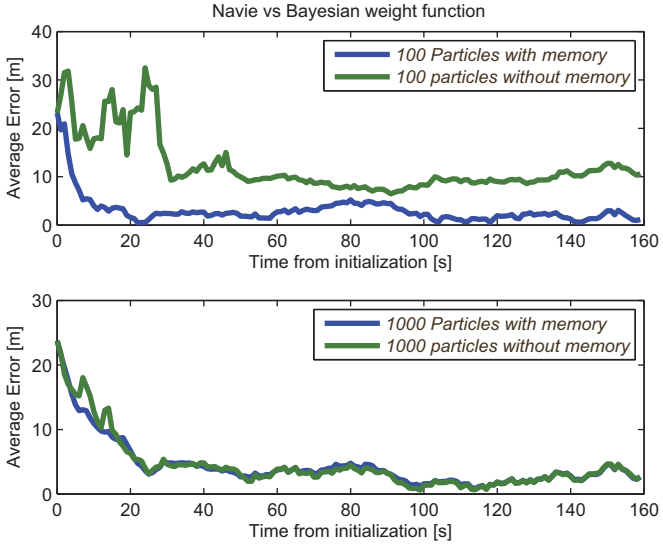

Figure 2. Naive versus Bayesian weight function. For small number of particles (100), the use of each particle history improves the solution and reduces the average error. For 1000 particles, the difference is unnoticed.

of particles (≈100 particles), significantly improves the results. This improvement, however, is barely noticed when the number of particles is higher (1000 particles).

While particle filter algorithms are nonparametric [19], they still demand for quite configurations and their efficiency heavily lies in an intelligent "sense" function. Evaluating such functions is not an easy task and one should be closely familiar with the domain to do so. A better, more accurate, "sense" function will yield more accurate results.

3.5. Simulation results of smartphone positioning

In this section, we present a set of simulations representing the presented particle filter in the setting of smartphone indoor navigation using floor-map constraints, RF positioning, and a pedometer. In **Figures 3–5**, we consider a "standard building" with a size of 10×20 m with about 10 rooms, the overall restriction area (i.e., walls) is about 10% of that area. The path is shown as a polygonal line, the real position is marked as a solid dark dot located on the path and the approximated position is marked as a lighter dot. The simulation uses the following

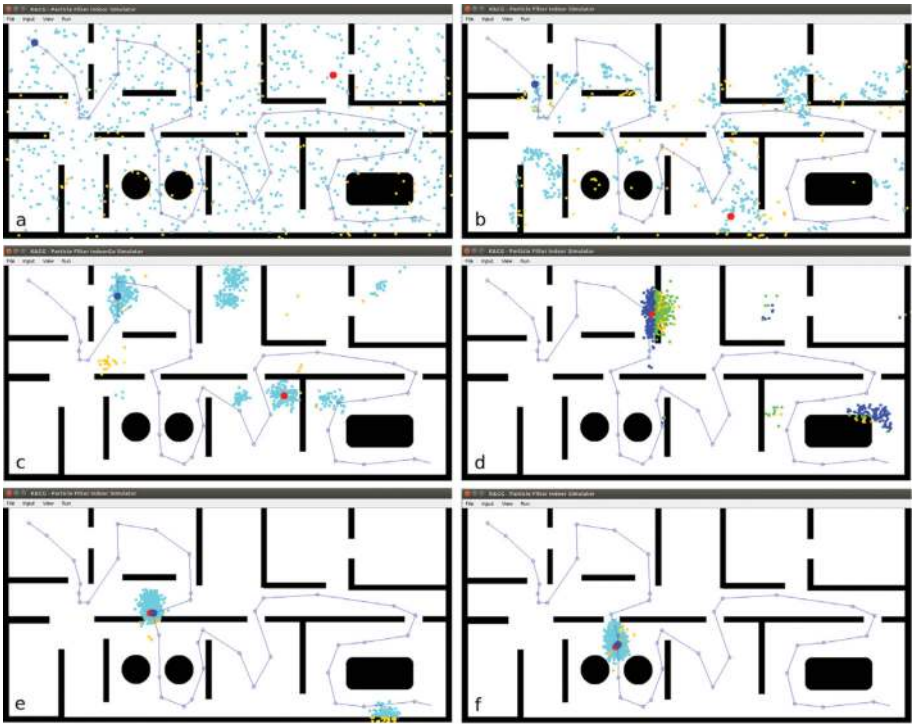

Figure 3. Particle filter in action: 800 particles were randomly located in the building (a) with no RF positioning service. Using pedometer combined with compass and a building floor map, the particles rapidly form few small clusters (b and c). (d and e) The correct solution (cluster) is computed yet the expected error is still relatively large—as there is a second (wrong) cluster of particles. (f) The algorithm converges and the expected accuracy is high.

Figure 4. Particle filter in action: 100 particles were randomly located in the building (a), a WiFi positioning service was used for fast converges (b and c). All the particles converged to the correct position (d).

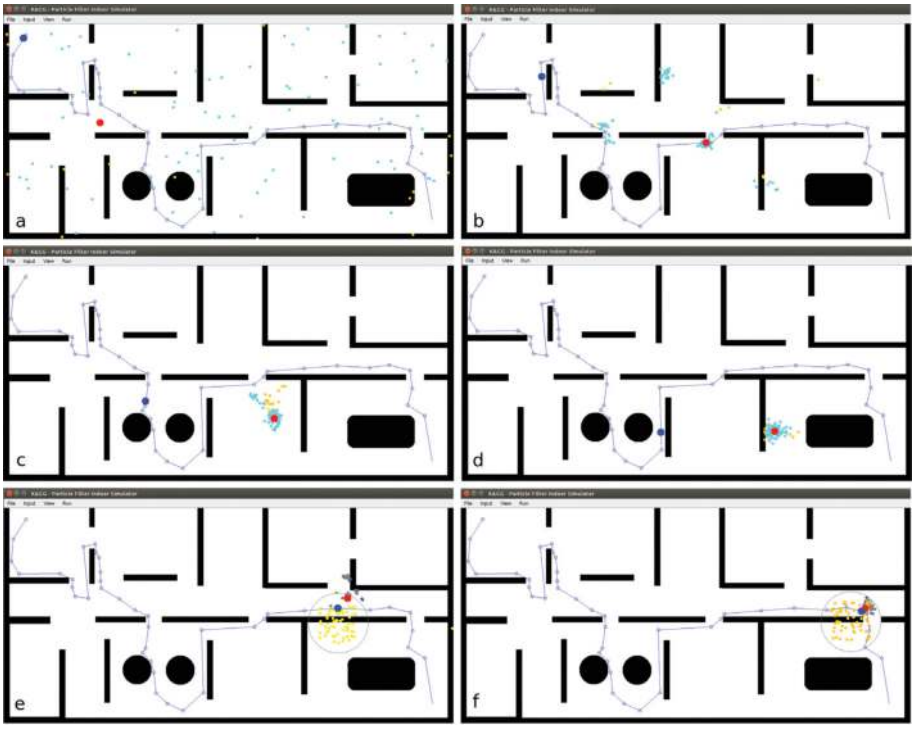

Figure 5. Particle filter in action: 100 particles were randomly located in the building (a). No RF positioning service is leading to a wrong converges (b–d), due to undersampling—the number of particles is too low. Using RF positioning (e and f), the correct position was found.

parameters: theWiFi expected position error is 5 m, the pedometer expected error is 20% in length, and 10 in angle.

In order to demonstrate the particle filter algorithm in the setting of indoor positioning, we first present the case (**Figure 3**) in which there is no RF (i.e., WiFi) positioning service. This case can also be seen in situations where the accuracy of the positioning service is larger than the region of interest. We then present (**Figures 4** and **5**) the more general case in which RF positioning service is available. Using this service, the particle filter converges rapidly even with significantly smaller set of particles.

3.6. 3D particle filter algorithm

Prior to this point, we have mainly considered the 2D case of the mobile phone indoor position algorithm. In this section, we will generalize the particle filter algorithm to a 3D case of multifloor buildings. In general, a building navigation is often referred to as 2.5D—in which the floor is assumed to be of a discrete value. Two modifications are needed in order to generalize the algorithm for 3D:

1. The floor(s) map should have an additional color (probability) representing the probability of floor change in each location: e.g., near by the elevator or the stairs, this probability (color) will be relatively high.
2. The action function should have an elevation (Δz) approximation. This functionality is commonly computed via a barometric pressure sensor. One should remember that these sensor are extremely sensitive to height differentiation (i.e., one cannot deduce the correct floor based on their value but one can confidently deduce floor change by inspecting the sensor's data derivative).

3.7. Overcoming the kidnapped robot problem

Since particle filters are not restricted to a single peak PDF, they can handle an ambiguous location scenario: e.g., two (almost) identical rooms in a hall. The filter maintains two clusters of particles in the candidate locations. Convergence is assumed when the receiver exits one of the rooms. Alas, a robust convergence to the true location is not guaranteed in a noisy environment, in particular, with a wrong map or biased sensors. Many indoor navigation algorithms suffer from this very problem (see **Figures 3** and **4** for such examples). City mall maps are occasionally changed and WiFi routers are moved from their previous location. These phenomena can throw a filter to converge to a wrong location (e.g., different floor). This is a major problem since standard particle filters hardly recover from a wrong position after they converge, in particular, since floor change is always accompanied with a major shift in the barometer sensor. Even if the receiver "wants" to converge to the true location, it cannot do so unless it is nearby an elevator or escalator. It is important to mention that this is a real problem in the realm of Inertial Navigation System (INS) and many algorithms will occasionally reset their value and start over. If one wishes to avoid such system resets, this issue must be addressed properly.

Wrong location convergence is very similar to another known problem, the kidnapped robot problem [19]. In the latter, we assume all the particles correctly converged to the true robot's position. However, a foe (intentional or not) kidnapped the robot and placed it outside the

convergence area. If no particle exists in the robot's new location, a true convergence is not possible. Therefore, a small portion of the particles (≈10% or less) is allocated in each phase to evenly respread in the ROI. Thus, the algorithm has a viable probability to reconverge to the true location. In our example, we evenly spread the particles in different floors.

4. Multiagents localization and tracking

Multitarget tracking (MTT) is a well-known problem in the field of image processing and estimation [23]. In particular, addressing the MTT problem utilizing particle filter techniques has been done previously [24, 25]. The scope of the problems MTT addresses is usually visual tracking. In this section, however, we would like to present a very easy-to-implement particle filter-based algorithm to tackle a nonvisual multiobject localization problem.

Section 2.2 presents several ways to report the estimated location based on the particles' distribution. As explained, all the methods assumed a single true solution, i.e., a single-agent localization. When two (or more) clusters of particles present, the algorithm either chooses the "best" solution from one of the clusters or attempts to average them all to produce a false answer somewhere in between them. When two (or more) clusters represent true position of several agents, we seek an algorithm which can both localize and track all of those agents.

4.1. Problem of interest

Our problem of interest consists of several radiant sources and several sensors which can detect this radiation. The aim is to localize, track, and estimate the numbers of radiant sources in the region. The radiation can be noise (sound waves), fire (heat or smoke), light, electromagnetic fields, etc. We chose to describe an interesting electromagnetic source—GNSS jammers.

4.1.1. GNSS jammers localization and tracking

The importance of GNSS is unquestionable. We rely on it more and more for both civilian and military uses. Attacking this system can cause a great deal of harm to any modern society. GNSS jammers jam the carrier frequency of the GNSS receiver (hence their name) by adding (transmitting) white noise to it. This process degrades the receiver signal-to-noise ratio (SNR) to a point where the receiver is unable to report its position, a phenomenon usually referred to as "losing fix." A 10 W jammer can paralyze GNSS receivers over a radius of a few hundred meters. Jamming interference can be detected by a degradation of the received satellites' SNR values. **Figure 6** demonstrates a typical jamming interference.

As **Figure 6** demonstrates, there exists a positive correlation between the receiver's SNR and its distance from the jammer. The black vertical line represents the jamming range; beyond that distance, the receiver is not affected by the jamming interference.

The reason why the jammer's location is interesting is twofold:

- First, the "losing fix" phenomena are similar to the uncertainty principle: when the receiver is far away from the jammer, it is not affected by it and produces a reliable GNSS

location. When the receiver is very close to the jammer, it cannot report its location since it has no fix.

- While it is easy to model a GNSS jammer as an omnidirectional antenna which degrades the received signals evenly in all directions, in reality, most jammers emit only in a certain direction.

As explained above, the problem of interest consists of both the radiant sources and the sensors. In our context, the emitting sources are the GNSS jammers, whereas the sensors are the GNSS receivers (e.g., smartphones) which can detect and report the satellites' received SNR. A strong SNR indicates no jamming interference. As the detected SNR decreases, the distance between the sensor and the jammer also decreases. This observation leads to the conclusion that if one can sample the SNR in the entire ROI, one can create a "heat map" of the most interfered points. Those points will serve as good candidates as the jammers' location. However, sampling the entire ROI is not viable in many cases, mainly due to map constraints. One needs to deduce the heat map form only a partial sampling.

4.1.2. Heat map

A plausible approach to tackle the emitting sources' problem is to define for each sensory record (sensor in time) a simplified probabilistic map (of the ROI) which represents the likelihood for the jammer to be at any point of the ROI. All the likelihood maps are combined one on top of the other to develop the heat map. **Figure 7** demonstrates such a heat map.

The experiment scenario is depicted in the left side of **Figure 7**. The red star represents the jammer's position. The yellow line represents the sensor's track. At each point, the sensor's SNR was recorded. As the sensor moves away from the jammer, its SNR increases and vice versa. Each such point creates a likelihood map. One can see that maximum intensity occurs in the vicinity of the real jammer.

When no jammer is presented, the maximum heat region will be outside the ROI, as expected. **Figure 8** demonstrates this phenomenon.

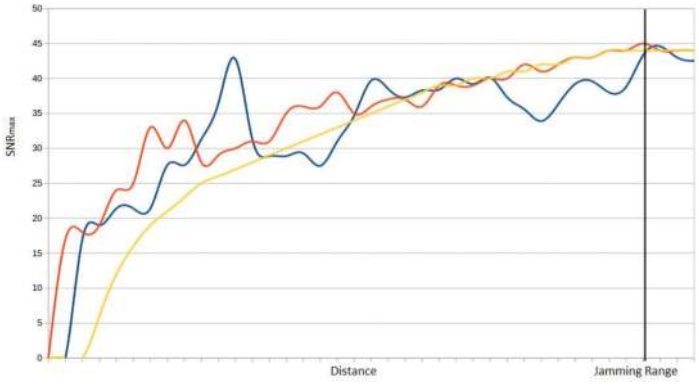

Figure 6. GNSS jamming SNR degradation. As the receiver approaches the jammer, its SNR decreases. As the receiver becomes farther away, its SNR increases. The yellow line represents the theoretical behavior, whereas the other colors represent real-world recoding figures.

This algorithm suffers from several inherent flaws:

- It assumes an omnidirectional pattern of the jammer. In other words, the algorithm cannot cope with more complex transmitting patterns.
- Since the algorithm has no notion of time, it is almost impossible to detect a nonstationary jammer.
- Much more important, the algorithm assumes the jammer's transmitting power is given. This, of course, is never the case in the real world.
- When two (or more) jammers are presented, the algorithm will not be able to differ between them (unless they are widely separated).

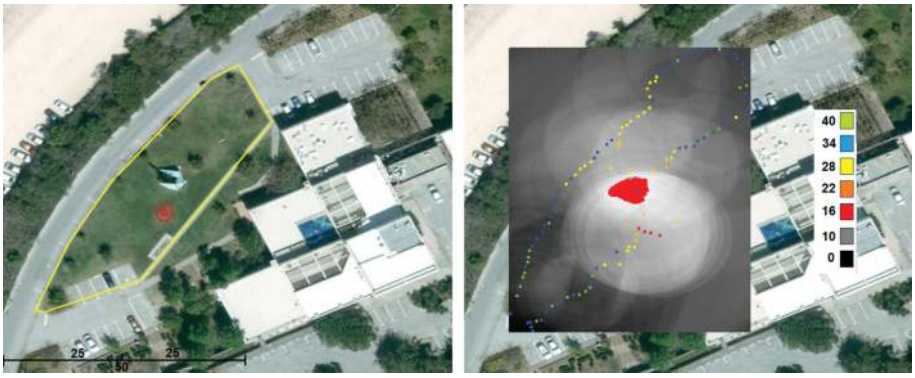

Figure 7. A heat map algorithm: Left: A field experiment in which a single jammer was located at the red star and the yellow polygon presents the sensor (GNSS receiver) path. Right: The samples of the sensor are presented in colored dots —the scale presents the maximal signal of the QNSS satellites. The heat map is presented in gray scale colors—the brightest region in the ROI is marked in red and overlaps the actual location of the jammer. In case of a single (fixed) jammer, this solution might be sufficient.

Figure 8. A heat map algorithm: No active jammer. Left: All samples. Right: The brightest regions of heat map are at the upper left and lower right corners. Since no jammer exists, the algorithm assumes it is outside the ROI.

Data: *ROI*, sensor data
Result: *GNSS* jammer location
1. Init: Distribute a set P of n particles in the *ROI*;
2. **while** *P is not converged* **do**
   3. **for each** *particle* $p \in P$ **do**
      4. Approximate the **action function** $u_t(p)$ from $\vec{v}_t$;
      5. Update p-position according to $u_t(p)$;
      6. Evaluate the belief (weight) function: $w(p)$ based on the sensors input.
   7. Resample P according the likelihood (weight) of each particle;
8. Compute best position in $P \in ROI$ as $pos(P)$;
9. **Report** $pos(P)$;

Algorithm 2: Single *GNSS* jammer tracking.

In order to tackle all of the above flaws, a more probabilistic approach should be taken. A (modified) particle filter can efficiently address the multiagent localization and tracking problem with a relatively small number of particles.

4.2. Single agent tracking algorithm

The following sections describe a robust method to detect, localize, and track several emitters in the ROI. For the sake of clarity, we first assume a single jammer scenario. Several jammers tracking algorithm will be explained in the next section. The proposed algorithm does not assume a known number of emitters or their exact transmitting power. Moreover, this algorithm copes well with scenarios where the jammers are mobile and may overlay each other. Since we seek to know the position (and velocity) of the jammer(s), each particle is defined as a possible jammer with a specific velocity, position, and transmitting power. The weight of each particle will be proportional to the number of sensors (smartphones) consistent with it. This approach assumes almost no prior knowledge regarding the jammers in the ROI. Since each particle holds a velocity vector, the algorithm can also track moving jammers. The formal description of the algorithm is given below:

As explained above, each particle represents a jammer. The initialization happens in line 1: each particle is a jammer with a different transmitting pattern. Its weight is proportional to the jammer's probability of being in a specific location. One can compute this probability as the sum of Gaussian distributions (since the pattern is known).

Convergence occurs when the longest distance between every two particles does not exceed a certain threshold. If no jammer exists in the ROI, the particles will not converge and the algorithm will not report a position. This algorithm tracks well a single jammer. However, the inherent nature of the particle filter prevents it from operating well in several jammers scenario. **Figure 9** demonstrates this problem. In this figure, one can see two jammers, each with a different strength and pattern, as represented by the black lines. The little squares represent several dozen sensors (smartphones). Although two jammers transmit in this scenario, all the particles converged to a single jammer.

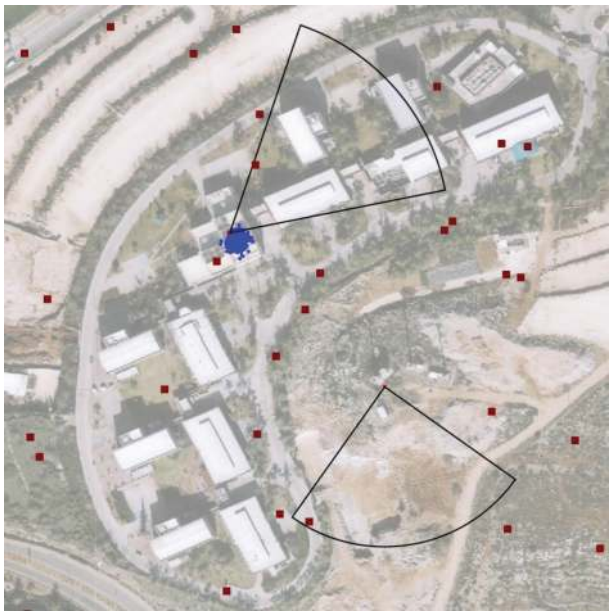

Figure 9. A single jammer localization. Although two jammers are active simultaneously, the particles converged only to one of them.

4.2.1. Unimodal versus multimodal

Although particles filters are not restricted to unimodal single peak PDFs, the resampling process itself tends to converges to a **single** value. Thus, two different clusters of particles (representing two jammers) will eventually converge to only one of the jammers due to the inherent resampling process. Should this was a typical clustering problem, a K-mean algorithm [26] would work well because the resampling process tends to favor one cluster over the others. This cluster will be the last one to survive and the other clusters (jammer) will be ignored. A typical particle filter algorithm can hold multimodal two-peak PDF, only as an intermediate phase. Multiagent tracking calls for a slightly different approach.

Data: ROI, sensor data
Result: *GNSS* jammers' locations
1. Init: Define *Jammers* to be an empty set of Jammers;
2. **while** *JammerDetection*(*ROI*) **do**
   3. Let J_i be *find*$_J$*ammer*(*ROI*);
   4. Add J_i to the *Jammers*;
   5. Update the ROI to be $ROI - ROI(J_i)$
6. return *Jammers*;

Algorithm 3: A generic algorithm for Multi Agent Localization.

4.3. Multiagent tracking algorithm

The multiagent tracking algorithm is very similar to Algorithm 2. The main difference is a more sophisticated approach toward convergence. After the algorithm converged to a single jammer, the algorithm respread particles outside the region of interference. The first set of particles is "assigned" to the first jammer and will track after it. The second set of particles will converge to another jammer relatively quickly. This happens due to the fact that the second set of particles is not affected by the first jammer. The formal description of this algorithm is given below.

Line 2: We denote *JammerDetection*(*ROI*). It is a Boolean function which returns true if the probability of having a jammer in the ROI exceeds a certain threshold.

Lines 3 and 4: Algorithm 2 was utilized to find the most probable jammer in the ROI. As mentioned above, Algorithm 2 reports a jammer position only after all the particles converged. If a jammer is detected, it will be added to the list (line 4).

Line 5: If a jammer was detected (line 3), the algorithm respreads particles outside the region of interference of the detected jammer. Calculating this region is easy since each particle holds the antenna pattern and the jamming transmitting power.

The next figures validate the algorithm's correctness. **Figure 10** depicts a two-jammer tracking scenario, each jammer having a different antenna pattern.

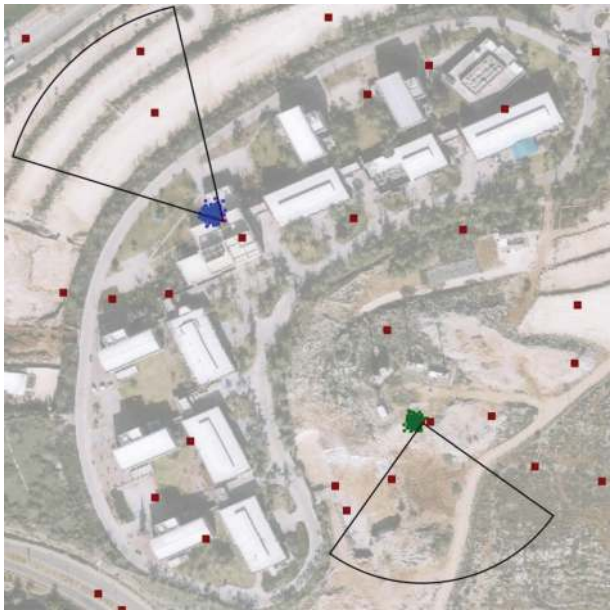

Figure 10. A single-jammer localization. Two jammers are active simultaneously and the algorithm converged to both of them.

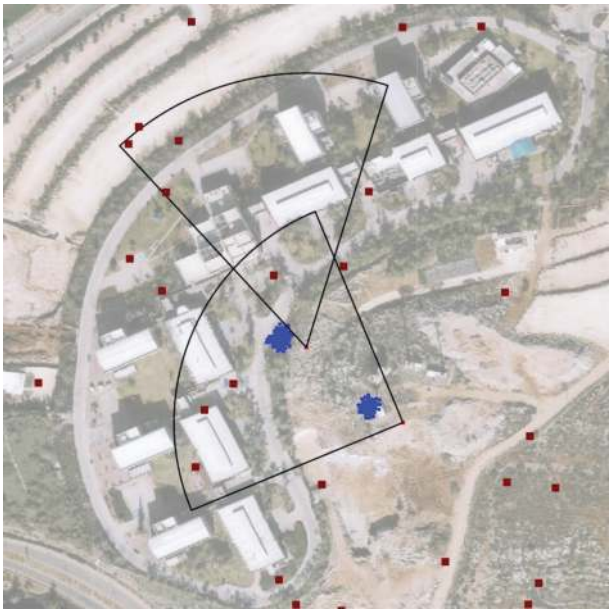

Figure 11. A single-jammer localization. Although two jammers are active simultaneously, the particles converged only to one of them.

Figure 11 shows an overlapping scenario. The interference regions of the jammers partly overlap and yet, the algorithm tracks each jammer separately.

4.4. Implementation remarks

The proposed framework for multiagent localization and tracking produces relatively accurate results, even with a small number of particles and sensors. While increasing the number of particles depends solely on computing power, increasing the number of sensors can be much more challenging and occasionally impossible. The results described here were achieved with 500 particles (for each jammer) and less than 30 sensors (smartphones).

5. Future research

Recent advances in technology such as autonomous driving, the Internet of Things (IoT), and bioinspired robotics require sophisticated and robust methods for computing probabilistic functions. In many cases, the problems of interest are NP-hard problems or have a real-time (or online) requirements and therefore cannot be solved accurately and efficiently using deterministic algorithms. Moreover, many mission critical systems are required to approximate not just the "state" (e.g., position and velocity) but suggest a tight bound for the expected error of the reported solution (i.e., an accuracy level). Such error-bound approximation is important for

autonomous platforms in which performing well in 99% of the time is insufficient. Heuristics based on particle filters allows a robust sensor fusion while maintaining the implementation relatively simple. Using such methods one can report the expected accuracy level (error). Using the modifications suggested in this work, one can significantly improve the expected runtime of a particle filter algorithm which makes it suitable even for real-time vision-based localization problems. For future work, we suggest that the need for robustness and real-time accurate results will require the use of massive parallel computation platforms such as GPU. Such platforms can allow an independent and parallel computing core for almost each particle and, therefore, to allow speedups of 10–100 times over existing solutions. Other research challenges include: designing particle filters on a sensor level—just as implementation of Kalman filter is common in embedded sensors. Finally, there is an important problem of setting and fine-tuning the parameters of a generic particle filter to a specific problem in the most suitable way. This research challenge can be seen as a double-stage particle filter: higher level particle filter seeks to improve its parameters while the lower level solves the problem of interest using a particle filter which uses the above parameters. Such a self-tuning heuristic might allow for a massive use of particle filter algorithms just as deep learning has allowed a greater and more efficient use of neural networks.

Author details

Roi Yozevitch* and Boaz Ben-Moshe

*Address all correspondence to: yozevitch@gmail.com

Ariel University, Ariel, Israel

References

[1] Hartmann K, Steup C. The vulnerability of UAVs to cyber-attacks—An approach to the risk assessment. In: 2013 5th International Conference on Cyber Conflict (CyCon). IEEE; 2013. pp. 1–23

[2] McKean HP. A class of Markov processes associated with nonlinear parabolic equations. Proceedings of the National Academy of Sciences. 1966;**56**(6):1907–1911

[3] Del Moral P. Non-linear filtering: Interacting particle resolution. Markov Processes and Related Fields. 1996;**2**(4):555–581

[4] Liu JS, Chen R. Sequential Monte Carlo methods for dynamic systems. Journal of the American Statistical Association. 1998;**93**(443):1032–1044

[5] Crisan D, Rozovskii B. The Oxford Handbook of Nonlinear Filtering. Oxford University Press, Oxford, England; 2011

[6] Flury T, Shephard N. Bayesian inference based only on simulated likelihood: Particle filter analysis of dynamic economic models. Econometric Theory. 2011;**27**(05):933–956

[7] Doucet A, De Freitas N, Gordon N. An introduction to sequential Monte Carlo methods. In: Sequential Monte Carlo Methods in Practice. Springer, Berlin, Germany; 2001. pp. 3–14

[8] Van Der Merwe R, Doucet A, De Freitas N, Wan E. The unscented particle filter. In: NIPS. Vol. 2000. Denver, CO, USA. 2000. pp. 584–590

[9] Gustafsson F. Particle filter theory and practice with positioning applications. IEEE Aerospace and Electronic Systems Magazine. 2010;**25**(7):53–82

[10] Nurminen H, Ristimaki A, Ali-Loytty S, Piché R. Particle filter and smoother for indoor localization. In: 2013 International Conference on Indoor Positioning and Indoor Navigation (IPIN). IEEE; 2013. pp. 1–10

[11] Montemayor AS, Pantrigo JJ, Sánchez Á, Fernández F. Particle filter on GPUS for real-time tracking. In: ACM SIGGRAPH 2004 Posters. ACM; 2004. p. 94

[12] Hendeby G, Hol JD, Karlsson R, Gustafsson F. A graphics processing unit implementation of the particle filter. In: 2007 15th European Signal Processing Conference. IEEE; 2007. pp. 1639–1643

[13] Chao M-A, Chu C-Y, Chao C-H, Wu A-Y. Efficient parallelized particle filter design on cuda. In: 2010 IEEE Workshop on Signal Processing Systems (SIPS). IEEE; 2010. pp. 299–304

[14] Farahmand S, Roumeliotis SI, Giannakis GB. Set-membership constrained particle filter: Distributed adaptation for sensor networks. IEEE Transactions on Signal Processing. 2011;**59**(9):4122–4138

[15] Rui Y, Chen Y. Better proposal distributions: Object tracking using unscented particle filter. In: Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2001. CVPR 2001., Vol. 2. IEEE; 2001. pp. 768–793

[16] Montemerlo M, Thrun S, Koller D, Wegbreit Ben, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. In AAAI/IAAI; 2002. pp. 593–598

[17] Petrovskaya A, Thrun S. Model based vehicle detection and tracking for autonomous urban driving. Autonomous Robots. 2009;**26**(2-3):123–139

[18] Niknejad HT, Takeuchi A, Mita S, McAllester D. On-road multivehicle tracking using deformable object model and particle filter with improved likelihood estimation. IEEE Transactions on Intelligent Transportation Systems. 2012;**13**(2):748–758

[19] Thrun S, Burgard W, Fox D. Probabilistic Robotics. MIT Press, Cambridge, Massachusetts (United States); 2005

[20] Gu Y, Lo A, Niemegeers I. A survey of indoor positioning systems for wireless personal networks. IEEE Communications Surveys & Tutorials. 2009;**11**(1):13–32

[21] Zekavat R, Buehrer RM. Handbook of Position Location: Theory, Practice and Advances. Vol. 27. John Wiley & Sons, Hoboken, New Jersey; 2011

[22] Honkavirta V, Perala T, Ali-Loytty S, Piché R. A comparative survey of WLAN location fingerprinting methods. In: 6th Workshop on Positioning, Navigation and Communication, 2009. WPNC 2009. IEEE; 2009. pp. 243–251

[23] Bar-Shalom Y. Multitarget-multisensor Tracking: Advanced Applications. Norwood, MA: Artech House; 1990. p. 391

[24] Hue C, Le Cadre J-P, Pérez P. Tracking multiple objects with particle filtering. IEEE Transactions on Aerospace and Electronic Systems. 2002;**38**(3):791–812

[25] Okuma K, Taleghani A, De Freitas N, Little JJ, Lowe DG. A boosted particle filter: Multitarget detection and tracking. In: European Conference on Computer Vision. Springer; 2004. pp. 28–39

[26] Hartigan JA, Wong MA. Algorithm as 136: A k-means clustering algorithm. Journal of the Royal Statistical Society. Series C (Applied Statistics). 1979;**28**(1):100–108